

JS Level 1



Введение



Объекты

В этой лекции мы с вами поговорим об объектах: возможности организовать удобное хранение и взаимодействие с данными. До этого мы рассматривали сугубо вычислительные задачи - нужно было что-то посчитать по формуле. Это нам было нужно, чтобы рассмотреть основы работы с языком. Сейчас мы возвращаемся к основной теме нашего курса – построению фронтенда для медиа-социальной сети.

Для этого нам нужно будет обсудить то, как организовать хранение данных (начнём с постов на странице сообщества) и как взаимодействовать с ними.



Объекты

Эта лекция может показаться сложной и, возможно, вам придётся перечитать её не один раз. Это нормально, в ней описываются достаточно сложные вещи, но мы потихоньку вас подготавливаем к тому, каково оно "быть программистом" в реальном мире – нужно уметь разбираться, в том числе читать документацию (чем вы и займётесь в рамках дополнительных ДЗ).



Повторение



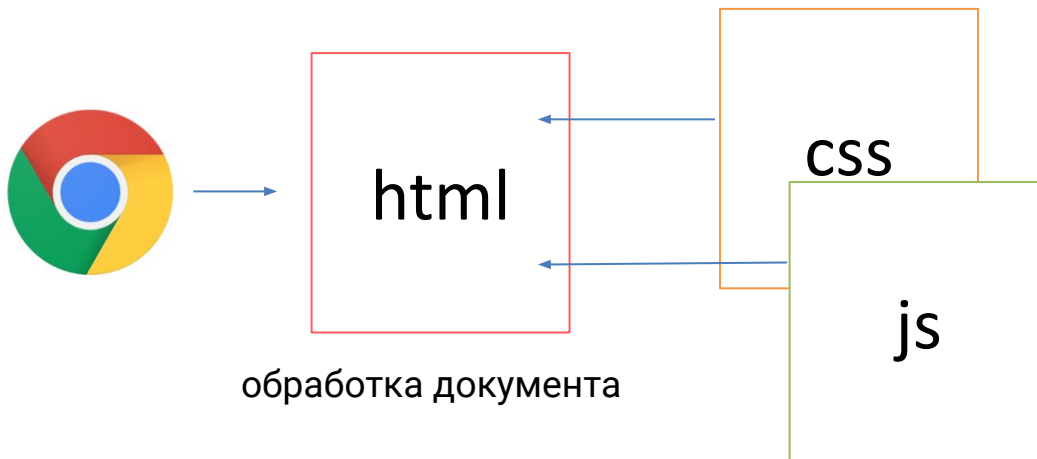
Web Application

На прошлых лекциях мы поговорили с вами, как работают веб-приложения (а именно их клиентская часть) – они загружаются и запускаются в браузере:



Ресурсы

Кроме того, мы обсудили сам механизм: сначала загружается HTML-документ (если вы указали его в адресной строке), а затем уже все ресурсы, которые в этом самом документе прописаны:



Ресурсы

После того, как ресурс загружен, он начинает обрабатываться браузером.

Например, браузер обрабатывал наш JS при каждой загрузке, выполняя код, расположенный в нём в соответствии с вызовами функций и условными операторами: браузер будет выполнять наши инструкции (напоминаю, мы пока учимся писать "вычислительную" часть нашей программы, но скоро перейдём к взаимодействию с интерфейсами).



Объекты



Объекты

Все типы данных делят на две большие категории:

1. Примитивы:

- a. `Boolean` – `true/false`
- b. `Null` – `null` (отсутствие значения*)
- c. `Undefined` – `undefined`
- d. `Number` – число
- e. `String` – строка
- f. `Symbol` – символы
- g. `BigInt` – большие целые числа

2. Объекты:

- a. `Object` – объект

Список типов данных не постоянен и может меняться (в новой версии добавлен тип для работы с большими числами: `BigInt`)

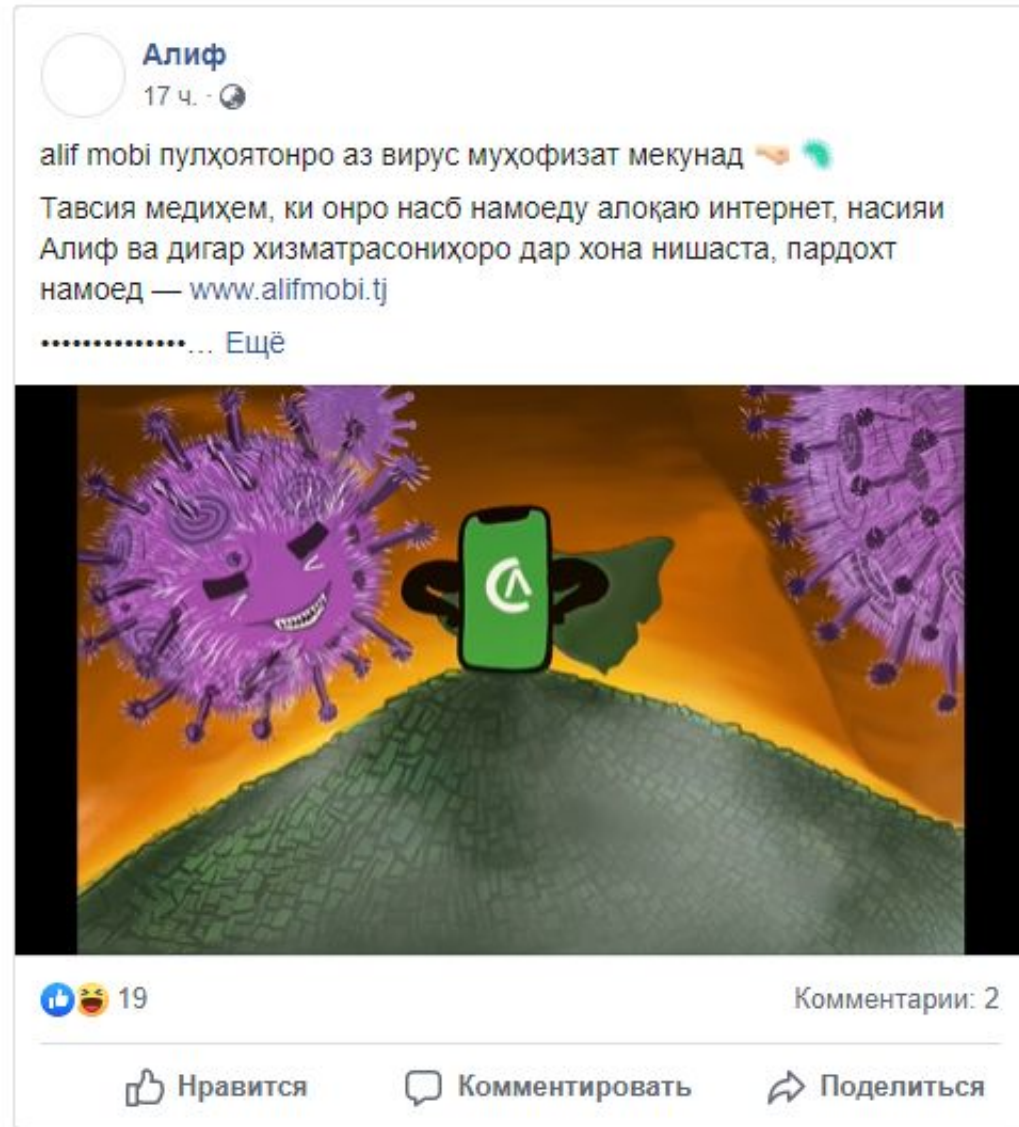


Задача

Возьмём пост из популярной соц.сети (см. скриншот справа) и попробуем подумать, что за информация там хранится.

Как вы видите, даже в одном посте очень много информации:

1. Логотип
2. Название
3. Время
4. Текст, видео
5. Лайки, комментарии и т.д.



Объекты

Обратите внимание: наш мозг устроен таким образом, что он группирует набор каких-то данных в одно понятие. Например, мы не говорим "Ты видел? В группе опубликовали интересный текст с интересным видео, датой публикации, логотипом и лайками". Мы говорим "Ты видел? В группе опубликовали интересный **пост**".

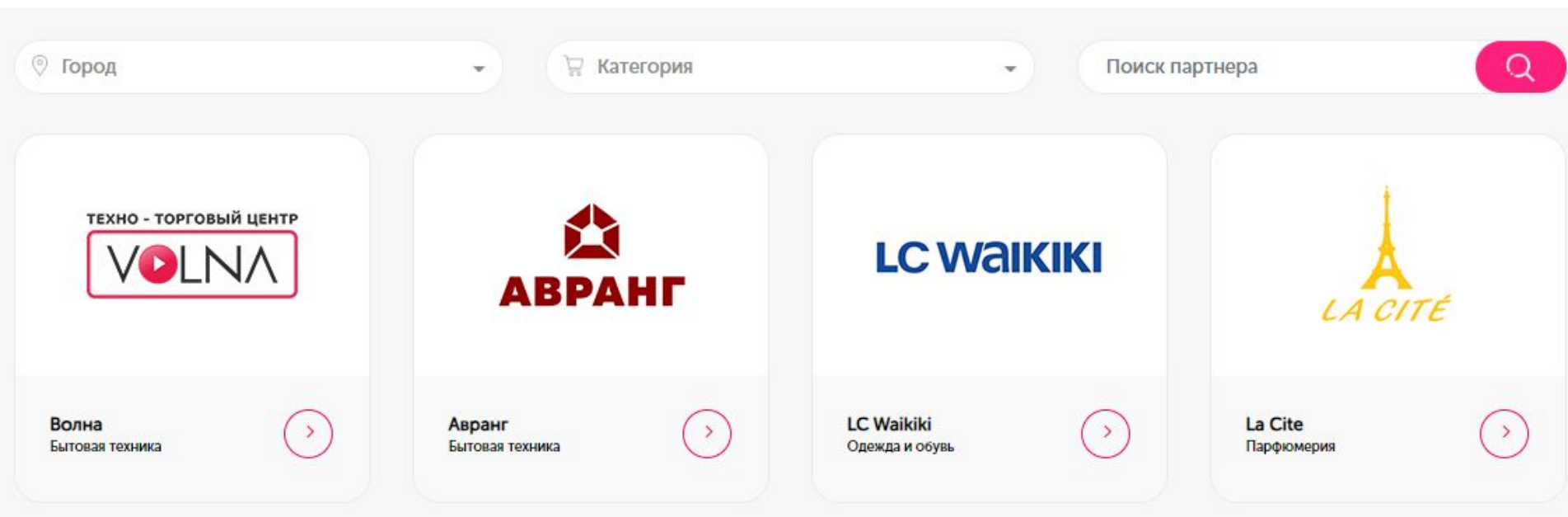
Т.е. мы просто придумываем для набора данных общее название, чтобы было удобнее общаться (точно так же мы для набора инструкций придумывали название в виде функции).

В реальной жизни всё точно так же: мы говорим "ты купил машину?", а не "ты купил колёса, двигатель, руль, салон, багажник, бензобак и т.д.?".



Задача

Если мы посмотрим, то ситуация схожая везде (даже не в соц.сетях всё примерно так же):



Объекты

Так вот объекты – это как раз набор данных, собранный в единую структуру, которой можно управлять как одним целым. Мы просто объединяем данные, которые бы пришлось хранить в разных переменных под одним именем:

```
1 const logoUrl = 'https://open.alif.academy/logo_alif.svg';  
2 const authorName = 'Алиф';  
3 const text = 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🗨️ 🟡...';  
4 const videoUrl = 'https://www.facebook.com/alifcapital/videos/675337203305709/';
```

Это и есть "Пост"

Раньше мы без этого прекрасно обходились, потому что наши задачи были сугубо вычислительные, но дальше без объектов никуда.



Объекты

Объект – это некая сущность (что-то, что мы можем выделить отдельно), обладающая набором свойств.

В реальной жизни объект чаще всего имеет какие-то физические границы: например, человек, автомобиль, здание, город, страна, планета. В мире программирования – объект это чаще всего какое-то понятие, обладающее свойствами. Например, пост или тарифный план:



Абонентская плата в месяц: 100 смн

- Пакет интернет трафика: **8000 МБ**
- Исходящие внутри сети: **Безлимит**
- Пакет минут по РТ: **100 минут**
- Пакет SMS по РТ: **800 SMS**

Команда подключения: *290*100#

Подключиться



Объекты

У объекта может быть миллион свойств (возьмём например, человека: имя, возраст, пол, вес, цвет глаз, должность, доход, счёт в банке и т.д.) и если мы попытаемся их все описать – ничего хорошего не выйдет (на самом деле мы зря потратим своё время).



Объекты

Давайте подумаем, как с этим обстоят дела в реальной жизни. Например, мы покупаем ноутбук, при этом нас чаще всего интересуют не все его свойства, а лишь небольшое количество:

1. Какой процессор, сколько ядер
2. Сколько памяти (ОЗУ)
3. Какого объёма жёсткий диск
4. Диагональ экрана
5. Вес и цвет
6. Время работы (без подзарядки)



Объекты

Что это значит? Это значит, что мы выбираем не все свойства объекта, а только самые важные для нашей программы. Например, в том же тарифном плане на сайте указано не всё, а только самое важное (в первую очередь, для покупателя):

И здесь действует всё тот же принцип: мы идём от простого к сложному - сначала выбираем минимальный набор свойств, а потом можем их добавлять, развивая нашу программу.



Абонентская плата в месяц: 100 смн

- Пакет интернет трафика: **8000 МБ**
- Исходящие внутри сети: **Безлимит**
- Пакет минут по РТ: **100 минут**
- Пакет SMS по РТ: **800 SMS**

Команда подключения: *290*100#

Подключиться

Абстракция

Это называется **абстракция** – мы выделяем и используем только то, что важно для решения нашей конкретной задачи.

Например, в рамках школы JS для нас важно количество решённых вами задач (и набранных баллов) и совсем не важны: пол, цвет глаз и возраст (главное, чтобы вам было больше 16), а также количество времени и попыток, которые вы затратили на решение задач.

Запомните: не стоит стремиться сразу написать идеальную программу, которая учитывает всё. Решите сначала самые простые задачи, а потом развивайте свою программу.



Объекты

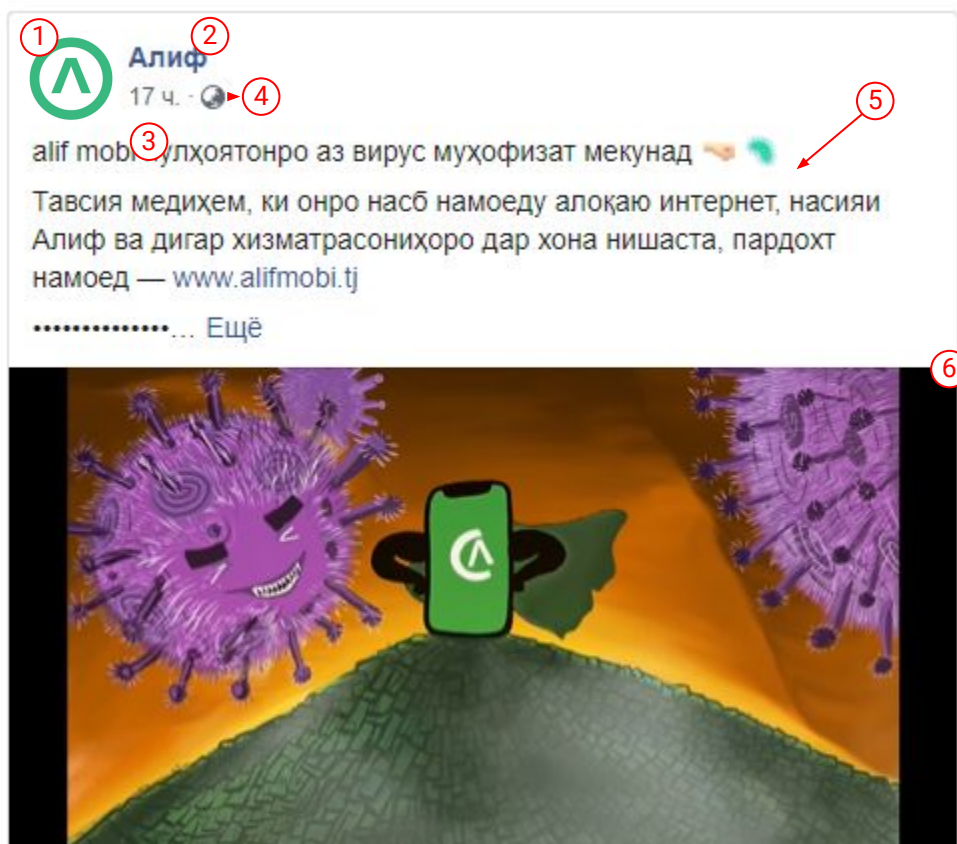
Вы не должны переживать по поводу того, что пока не совсем понятно, что такое свойства объекта и какие они должны быть, потому что вариантов на самом деле всего 4:

1. Вы делаете аналог какого-то сервиса (например, соц.сеть) и смотрите, как сделали другие
2. Вам уже приносят готовый набор свойств (например, в тех же мобильных операторах, тарифы придумывают специальные люди и говорят вам, что и как нужно отображать на сайте)
3. Вам дают уже готовые объекты, которыми вы можете пользоваться (например, создатели браузера предоставляют вам эти объекты, нужно лишь научиться ими пользоваться)
4. Вы придумываете всё сами (такое бывает достаточно редко, чаще всего этот вариант – это комбинация предыдущих + анализ вашей задачи)



Свойства

Давайте вернёмся к посту в Facebook и попробуем отработать метод выделения свойств объекта. Метод этот очень прост – мы смотрим на объект и анализируем его сверху-вниз, слева-направо, записывая всё в свойства:



1. Логотип
2. Имя автора
3. Сколько часов назад опубликовано (или когда создано)
4. Общедоступно или нет
5. Текст
6. Видео
7. Ниже лайки и т.д.




Свойства

А дальше мы просто выбираем нужные типы для хранения этих данных, например:

```
const logoUrl = 'https://alif-skills.pro/media/alif-logo.svg';  
const authorName = 'Алиф';
```

Значения в одинарных кавычках – это строки (строки могут ещё писаться в `""` и ```).

В данном случае `url` – это просто ссылка, по которой браузер может загрузить картинку:  (работает это так же, как с CSS: браузер встречает ссылку на ресурс, загружает его, обрабатывает и отображает).

Имя автора – тоже строка (потому что ни число, ни `undefined`, ни `boolean` сюда не подходят).



Время

Дальше вопрос со временем: как хранить время? Для этого нужно внимательно посмотреть на посты:

48 мин. · 🌐 2 ч. · 🌐 21 ч. · 🌐 24 апреля в 15:47 · 🌐

Как же это хранить? Вперемешку минуты, часы и даты. Это первый вопрос.

Второй вопрос: есть такая вещь как часовые пояса. 12 часов дня в Душанбе != 12 часам дня в Москве, и если вы строите продукт с расчётом на международный рынок, это нужно учитывать.



Время

На самом деле, в качестве отправной точки достаточно часто берётся время по UTC (всемирное координированное время), а именно дата 1 января 1970 года 00:00:00 и считается количество секунд (или миллисекунд), прошедшее с этой даты (оно одинаково вне зависимости от часового пояса).

А количество секунд – это просто число. Отметки же 48 минут и т.д. просто преобразование этой разницы с помощью JS.

```
const created = 1588166860;
```

Называют эту величину – unix time.



Общедоступно или нет

На самом деле, здесь тоже нужно посмотреть, какие ещё возможны варианты, но пока мы остановимся на том, что пост может быть либо общедоступным (видно всем), либо нет.

```
const public = true;
```

А если значения всего два из разряда да/нет, то автоматически напрашивается **boolean**. А если планируется несколько, то лучше сразу строкой.



Текст и видео

С текстом и видео, в принципе, всё просто:

1. Текст – это строка
2. Видео – это ссылка

```
const text = 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🗨️ 🦠...';  
const videoUrl = 'https://www.facebook.com/alifcapital/videos/675337203305709/';
```

Единственное замечание, которое здесь есть – это замечательные символы, которые называются Emoji. С ними нужно быть достаточно аккуратным (и иногда лучше использовать картинки), потому что на разных устройствах они могут отображаться по-разному (а где-то вообще не отображаться):

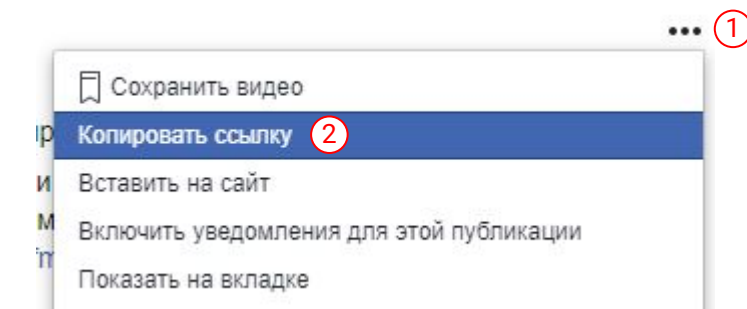


id

На самом деле, почти у каждого объекта, в котором хранятся какие-то данные (например, пост, тариф или ещё что-то), есть ещё одно свойство, про которое вы обязательно должны помнить. Нажмите на три точки (1) в правом верхнем углу и выберите скопировать ссылку (2):

Ссылка будет вот такая:

<https://www.facebook.com/alifcapital/videos/675337203305709/>



id



id – это число, либо строка, которая уникальным образом позволяет найти объект определённого вида в системе. Например, для человека – это может быть серия + номер паспорта, а для организации – ИНН.

id



У другого видео: <https://www.facebook.com/alifcapital/videos/686174348801048/>



Как мы узнали?

Как мы узнали про `id` и про время в формате `unix time`? На самом деле, вариантов всего два:

1. Почитать документацию на API (что это такое мы узнаем чуть позже)
2. Посмотреть исходники страницы и передаваемые данные (это вы научитесь делать, если дойдёте до следующих лекций)

На самом деле, через какое-то время вы будете добавлять `id` и дату создания почти во все объекты (и привыкнете к этому)



Объекты

Остальное мы для простоты опустим, и попробуем теперь взглянуть на то, что получилось:

```
const logoUrl = 'https://alif-skills.pro/media/alif-logo.svg';
const authorName = 'Алиф';
const created = 1588166860;
const public = true;
const text = 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🍷 🦠...';
const videoUrl = 'https://www.facebook.com/alifcapital/videos/675337203305709/';
```

Соберём это всё в объект и обсудим то, как с этим работать:

```
const post = {
  logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф',
  created: 1588166860,
  public: true,
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🍷 🦠...',
  videoUrl: 'https://www.facebook.com/alifcapital/videos/675337203305709/',
};
```



Объекты

Та форма записи объекта записи объекта, которую мы вам показали – литеральная: мы пишем имя переменной, оператор присваивания и дальше в фигурных скобках перечисляем набор свойств, которые есть у этого объекта:

```
const post = {  
  logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',  
  authorName: 'Алиф',  
  created: 1588166860,  
  public: true,  
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🍌 🦠...',  
  videoUrl: 'https://www.facebook.com/alifcapital/videos/675337203305709/'  
};
```

ставится для удобства
добавления новых
свойств

Обратите внимание: эти фигурные скобки – это не блок кода, это форма записи объекта. В конце ставится точка с запятой. Свойства пишутся в формате:

имяСвойства: значение,

Требования к именам свойств такие же, как к именам переменных.



Литерал

Что такое литерал? Литерал это непосредственное значение, записанное в коде.

Примеры:

- `const percent = 0.007;` // 0.007 и есть литерал (числовой)
- `const isRegistered = false;` // false и есть литерал (boolean)
- `const message = 'worked!';` // 'worked!' и есть литерал (строковый)

Мы увидели с вами литерал объекта на предыдущем слайде (а ближе к концу лекции познакомимся с литералом массива).



API



API

Шагнём немного в сторону и обсудим термин API (Application Programming Interface) – возможность двух программ взаимодействовать друг с другом. Это важно! Не человека с программой, а именно двух программ между собой.

Большие сервисы, например, соц.сети, мессенджеры, банки предоставляют API, чтобы вы могли писать программы, взаимодействующие с их программами.

В случае соц.сетей – это просматривать/публиковать посты, в мессенджерах – писать ботов, в банках – проводить платежи.



API

Почему это важно для нас с вами? Потому что вы можете изучать документацию (описание того, как всё устроено и как пользоваться) на чужое API и перенимать опыт: понимать, как разработчики решали те или иные проблемы, какие объекты они создавали и т.д.



API

Например, API Facebook/Meta на посты расположено на странице <https://developers.facebook.com/docs/graph-api/reference/post/>. Оно описано не очень хорошо, но позволяет понять, как строятся объекты.

То же самое для Vk: <https://vk.com/dev/wall.get> (гораздо удобнее и на русском).

Важно: уделяйте время тому, чтобы изучать, как проектируются "большие" системы. Это поможет вам лучше проектировать свои.

Примечание: не всегда чужие системы спроектированы идеально, но даже с учётом этого вам необходимо расширять кругозор и смотреть, кто и как сделал.



Работа со свойствами



Работа со свойствами

Итак, создавать объект мы научились, давайте попробуем поработать со свойствами:

```
1 const post = {  
2   logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',  
3   authorName: 'Алиф',  
4   created: 1588166860,  
5   public: true,  
6   text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🍌 🍌 ...',  
7   videoUrl: 'https://www.facebook.com/alifcapital/videos/675337203305709/',  
8 };  
9  
10 console.log(post.authorName);  
11  
12 post.authorName = 'Алиф Банк';
```

чтение свойства

запись свойства



Работа со свойствами

На 10-ой строке мы обращаемся к свойству объекта: сначала пишется имя переменной, в которой хранится объект, затем оператор `.` (обращение к свойству) и имя свойства:

`объект.свойство` // чтение свойства объекта

На 12-ой строке мы назначаем свойству объекта новое значение: также сначала пишется имя переменной, в которой хранится объект, затем оператор `.`, имя свойства и присваивание `=`:

`объект.свойство = значение` // запись в свойство объекта нового значения



Работа со свойствами

Таким образом, оператор `.` позволяет обращаться к конкретному свойству объекта.

Важно: не путать с точкой в числе, например, `3.14`, здесь точка – это разделитель целой и дробной части.



Свойства

Как это запомнить? Представим, что у вас есть друг Вася:

```
const vasya = {  
  age: 17,  
  ... // другие свойства  
};
```

Когда вы к нему обращаетесь с вопросом, то вы делаете это примерно следующим образом:

"Вася, сколько тебе лет?"

vasya.age


То же самое относится к записи свойств.



Свойства


Давайте попробуем вывести объект в консоль:

```
> post
< {logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф Банк', created: 1588166860, public: true,
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекуна
  Д 🍌 🌱...', ...}
```



Не совсем удобно, но можно нажать на треугольную стрелку и развернуть объект:

```
> post
< {logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф Банк', created: 1588166860, public: true,
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекуна
  Д 🍌 🌱...', ...} ⓘ
  authorName: "Алиф Банк"
  created: 1588166860
  logoUrl: "https://alif-skills.pro/media/alif-logo.svg"
  public: true
  text: "alif mobi пулҳоятонро аз вирус муҳофизат мекунад"
  videoUrl: "https://www.facebook.com/alifcapital/videos/6"
  ▶ [[Prototype]]: Object
```



Так уже лучше, хотя есть какое-то непонятное свойство `[[Prototype]]`, которое мы с вами не создавали и не объявляли (к нему мы вернёмся на следующей лекции).



__proto__

У некоторых из вас вместо `[[Prototype]]` будет написано `__proto__`. На текущий момент это не критично.



Свойства

JS достаточно мощный язык и он позволяет вам "на лету" добавлять свойства в объект, а также удалять их. Например, мы можем в любой момент добавить свойство `likes` (количество лайков):

```
> post
< {logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф Банк', created: 1588166860, public: true,
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад',
  videoUrl: 'https://www.facebook.com/alifcapital/videos/6111111111111111'}
  authorName: "Алиф Банк"
  created: 1588166860
  logoUrl: "https://alif-skills.pro/media/alif-logo.svg"
  public: true
  text: "alif mobi пулҳоятонро аз вирус муҳофизат мекунад"
  videoUrl: "https://www.facebook.com/alifcapital/videos/6111111111111111"
  ► [[Prototype]]: Object

> post.likes = 50; // добавляем свойство
< 50

> post.likes; // выводим добавленное свойство
< 50
```



Свойства

После этого оно появится в самом объекте:

```
> post.likes = 50; // добавляем свойство
< 50

> post.likes; // выводим добавленное свойство
< 50

> post
< {logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф Банк', created: 1588166860, public: true,
  text: 'alif mobi пулхоятонро аз вирус муҳофизат мекунад 🍌 🌱...', ...} ⓘ
  → likes: 50
  logoUrl: "https://alif-skills.pro/media/alif-logo.svg"
  public: true
  text: "alif mobi пулхоятонро аз вирус муҳофизат мекунад"
  videoUrl: "https://www.facebook.com/alifcapital/videos/"
  ► [[Prototype]]: Object
```



Свойства

Так же, как и добавили, мы можем удалить его с помощью оператора **delete**:

```
< {logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф Банк', created: 1588166860, public: true,
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🍌 🍌...', ...}
  authorName: "Алиф Банк"
  created: 1588166860
  likes: 50
  logoUrl: "https://alif-skills.pro/media/alif-logo.svg"
  public: true
  text: "alif mobi пулҳоятонро аз вирус муҳофизат мекунад"
  videoUrl: "https://www.facebook.com/alifcapital/videos/"
  ► [[Prototype]]: Object
```

```
> delete post.likes ←
```

```
< true
```

```
> post
```

```
< {logoUrl: 'https://alif-skills.pro/media/alif-logo.svg',
  authorName: 'Алиф Банк', created: 1588166860, public: true,
  text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🍌 🍌...', ...}
  authorName: "Алиф Банк"
  created: 1588166860
  logoUrl: "https://alif-skills.pro/media/alif-logo.svg"
  public: true
  text: "alif mobi пулҳоятонро аз вирус муҳофизат мекунад"
  videoUrl: "https://www.facebook.com/alifcapital/videos/"
  ► [[Prototype]]: Object
```



Добавление/удаление СВОЙСТВ

И то, и другое (и добавление, и удаление) достаточно опасные операции и могут сыграть с вами злую шутку: вы можете случайно добавить не то свойство или удалить нужное, а потом потратить часы на отладку.

Поэтому лучше не добавлять и не удалять свойства без лишней необходимости после создания объекта*.

Примечание*: это не железное правило и мы посмотрим, где и когда можно попробовать его нарушить.



const

Вы можете сказать "но мы же говорили, что `const` – не позволяет изменять связку между именем и значением?". Всё дело в том, что `const` – не позволяет менять именно связь между именем и значением, но позволяет менять свойства объектов.

Это как с вашим другом – сам человек не меняется (т.е. физически, это тот же самый человек), но его свойства могут поменяться, например, он может сменить причёску, фамилию и т.д.



Свойства

Нужно отметить, что свойствами объекта не обязательно могут быть только значения примитивных типов.

Вполне допустимо в качестве свойств объектов использовать другие объекты, таким образом организовывая очень сложные объекты (как и в реальном мире).

Например, если мы посмотрим на автомобиль, то это объект, но он в свою очередь состоит из других объектов (например, двигатель – это тоже объект, который тоже может состоять из других объектов).



Web API



console

Интересно, заметили вы или нет, что когда мы работаем с консолью (например, выводим что-то туда), то вызов выглядит следующим образом:

```
console.log(...);
```

Очень похоже на то, как мы обращались к свойствам объекта, не так ли? Но в то же время, мы с вами не создавали объект и клали его в переменную с именем `console`.



console

На самом деле, это именно браузер предоставляет нам готовый объект `console`, который уже обладает рядом свойств.

Давайте попробуем его распечатать.



console

Интересно, т.е. это объект, но его свойства помечены буквой **f** и круглыми скобками, что это значит?

```
> console
< console {debug: f, error: f, info: f, log: f, warn:
  f, ...} ⓘ
  ▶ assert: f assert()
  ▶ clear: f clear()
  ▶ context: f context()
  ▶ count: f count()
  ▶ countReset: f countReset()
  ▶ createTask: f createTask()
  ▶ debug: f debug()
  ▶ dir: f dir()
  ▶ dirxml: f dirxml()
  ▶ error: f error()
  ▶ group: f group()
  ▶ groupCollapsed: f groupCollapsed()
  ▶ groupEnd: f groupEnd()
  ▶ info: f info()
  ▶ log: f log()
  ▶ memory: MemoryInfo {totalJSHeapSize: 100000000, usedJSHe
  ▶ profile: f profile()
  ▶ profileEnd: f profileEnd()
  ▶ table: f table()
  ▶ time: f time()
  ▶ timeEnd: f timeEnd()
  ▶ timeLog: f timeLog()
  ▶ timeStamp: f timeStamp()
  ▶ trace: f trace()
  ▶ warn: f warn()
    Symbol(Symbol.toStringTag): "Object"
  ▶ [[Prototype]]: Object
```

Функции

В JS функции это тоже объекты. Пока звучит немного странно, но вы должны к этому привыкнуть. Ключевая идея: функция – это объект, который ко всему прочему можно вызывать (тогда выполнится код внутри функции).

Поскольку функция – это объект, мы можем её положить в свойство другого объекта (как и любой другой объект)



Функции

Когда функция является свойством какого-то объекта вы можете воспринимать это следующим образом: есть смартфон, у него есть функция "позвонить". Эта функция принадлежит конкретному объекту. Т.е. не можете просто "позвонить", если у вас нет смартфона, с которого можно звонить.

Такие функции (которые лежат в свойствах объектов), называются методами. Пока нам не нужно уметь создавать их, важно сначала научиться ими пользоваться.

Но при этом вполне можно положить в свойство объекта "обычную" функцию, которой не нужен объект для работы.



Промежуточные итоги



Промежуточные итоги

Мы с вами научились создавать объекты, взаимодействовать с их свойствами.

Кроме того, поговорили о том, что свойствами объектов могут быть функции (тогда их называют методами) и что браузер предоставляет нам ряд готовых объектов (пока обсудили только `console`), которые нам не нужно самостоятельно создавать, мы их можем сразу же использовать.



Промежуточные итоги

Это всё хорошо, но пока мы научились создавать только один объект, который описывает ровно один пост. Но постов на странице много. Как нам их хранить? И как с ними работать?

Мы же не можем создавать переменные `post1`, `post2`, `post3`, ..., `postN` (за это бот отправит вас на доработку), тем более мы не знаем точного количества постов (они каждый раз подгружаются, когда мы пролистываем до низу). Что же делать?



Массивы



Массив

Массив – это специальный тип объектов, позволяющий удобно управлять набором данных. Этот тип один из ключевых в вашей работе, поэтому уделите должное внимание его использованию.

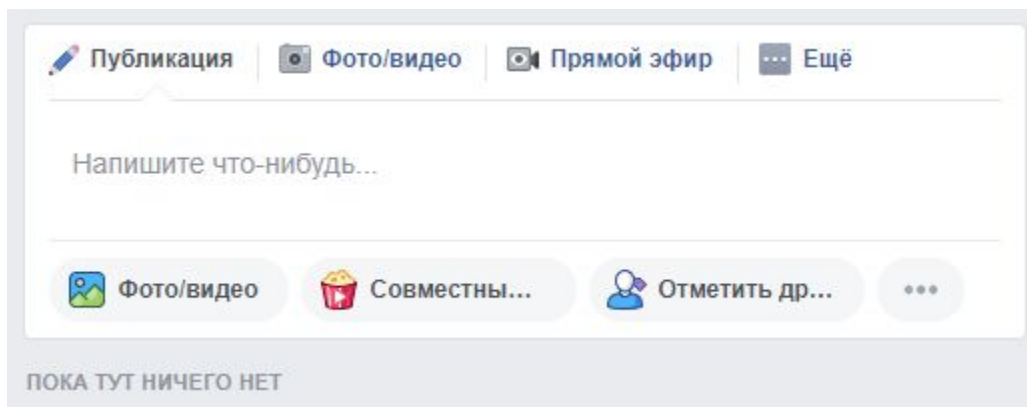
Можете представлять себе массив как список элементов со следующими характеристиками:

1. Все элементы расположены упорядоченно (т.е. у каждого элемента есть своё место)
2. Элементы могут быть разного типа
3. Элементы можно добавлять в список/удалять/заменять
4. Элементы можно перебирать по одному



Массив

Давайте попробуем воспользоваться массивами. Сначала научимся создавать пустой массив элементов (представим, что мы только создали новую группу в соц. сети и там нет никаких записей):



Рекомендуем вам создать новый проект и делать в нём

```
const posts = [];
```

Обратите внимание, что переменные для массивов стараются называть во множественном числе, чтобы показать, что этот объект хранит именно "набор постов".



Массив

Попробуем добавить в массив несколько постов (для сокращения кода мы будем указывать не все поля из тех, что обсуждали):

```
const posts = [  
  {  
    id: 2,  
    authorName: 'Алиф',  
    created: 1588235339,  
    text: 'Как отправлять деньги из России в Таджикистан без комиссии?',  
  },  
  {  
    id: 1,  
    authorName: 'Алиф',  
    created: 1588166860,  
    text: 'alif mobi пулҳоятонро аз вирус муҳофизат мекунад 🗨️ 🦠...',  
    likes: 10,  
  },  
];
```

не обязательно давать
объектам имена
можно сразу хранить в
массиве



Массив

Если мы посмотрим в консоль, то окажется, что созданный нами массив – достаточно умный объект, он сам умеет считать, сколько в нём элементов:

```
> posts
< ▼ (2) [{...}, {...}] ⓘ
  ▶ 0: {id: 2, authorName: 'Алиф', created: 1588235339, text
  ▶ 1: {id: 1, authorName: 'Алиф', created: 1588166860, text
    length: 2
  ▶ [[Prototype]]: Array(0)
>
```

Но самое интересно заключается в следующем: при выводе в консоль, массив очень похож на обычный объект, вот только свойства его "называются" числами, причём начиная с 0.

Первый вопрос, который должен возникнуть: теперь у наших постов нет имён (мы не сохранили их в переменных), как же нам до них добраться?



Доступ к свойствам

На самом деле, для того, чтобы получить доступ к свойствам объектов, существует три формы записи:

1. `имяПеременной.имяСвойства` – если имя свойства соответствует правилам для имён переменных
2. `имяПеременной[имяСвойства]` – если имя свойства число или хранится в переменной
3. `имяПеременной['имяСвойства']` – универсальный способ (имя свойства может быть любым, включая варианты для п.1 и п.2)



Доступ к свойствам

0 и 1 – не являются допустимыми именами для переменных, поэтому вариант 1 мы использовать не можем, а вот варианты 2 и 3 – вполне:

```
> posts[0] ← Предпочтительный вариант для массивов
< {id: 2, authorName: 'Алиф', created: 1588235339, text: 'К
  ► ак отправлять деньги из России в Таджикистан без комисси
  и?'}
```

```
> posts['0']
< {id: 2, authorName: 'Алиф', created: 1588235339, text: 'К
  ► ак отправлять деньги из России в Таджикистан без комисси
  и?'}
```

Таким образом, мы можем не придумывать имена переменных для каждого поста, а просто хранить все посты в массиве и дать имя только массиву (тем более нормальных имён мы им всё равно не сможем придумать).



Индексы

Вот эти числовые свойства в массиве называются индексами и нумеруются последовательно, начиная с 0. Это важно запомнить, что нумерация происходит автоматически и у первого элемента индекс 0.

Давайте посмотрим, что будет, если мы попробуем обратиться к несуществующему индексу:

```
> posts[99]  
< undefined
```

Нужно запомнить это – JS возвращает `undefined`, при попытке доступа к несуществующему свойству, не важно, какой из трёх способов для доступа к свойству вы используете:

```
> posts.unknown  
< undefined
```



Циклы

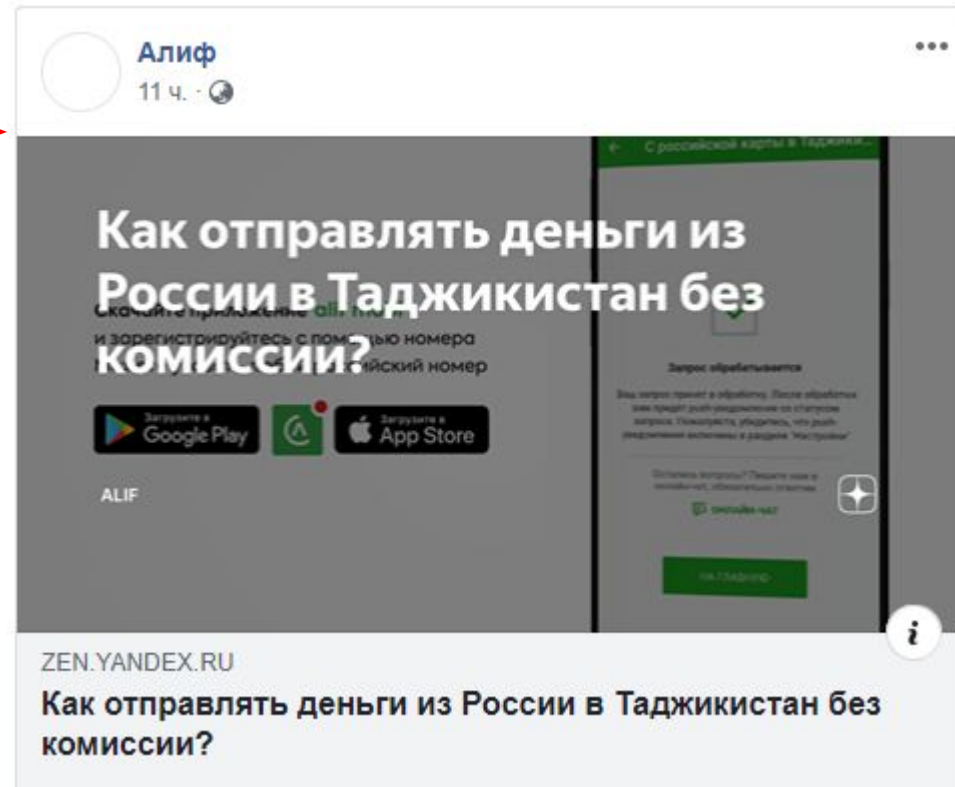


Циклы

Это всё хорошо, но мы же не можем в программу "зашить" фиксированные индексы элементов. Ведь мы не знаем сколько их будет. По-хорошему, нам нужно сделать нечто следующее: для каждого элемента из массива нарисовать "карточку" на веб-странице*:

```
> posts
< ▼ (2) [{...}, {...}] ⓘ
  ▶ 0: {id: 2, authorName: 'Алиф', created:
  ▶ 1: {id: 1, authorName: 'Алиф', created:
    length: 2
  ▶ [[Prototype]]: Array(0)
```

Примечание*: в современных системах это так и происходит (элементы на странице создаются средствами JS).

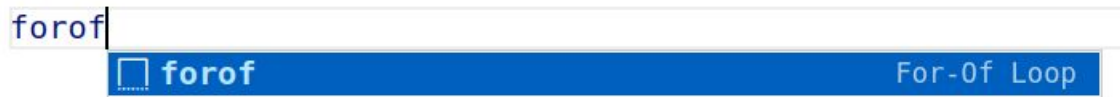


Цикл

Цикл – это конструкция, позволяющая многократно выполнять определённый набор действий. Например, мы сделаем пока самую тривиальную вещь: для каждого поста выведем в консоль текст поста (свойство `text`).


В JS есть несколько циклов, мы рассмотрим с вами их достаточно поверхностно, поскольку практически не будем использовать.

В VS Code наберите `forof` и нажмите `Tab`:



Цикл

Появится вот такая конструкция, в которой `iterator` надо заменить на `post`, а `object` на `posts`:

```

for (const iterator of object) {
  |
}
```



```
for (const post of posts) {
  | console.log(post.text);
}
```

В консоли увидим:

Как отправлять деньги из России в Таджикистан без комиссии?	<code>app.js:18</code>
--	------------------------

alif mobi пулхоятонро аз вирус муҳофизат мекунад 🗨️ 🌿 ...	<code>app.js:18</code>
--	------------------------



Цикл

Самостоятельно поставьте точку остановки на внутри цикла и убедитесь, что браузер каждый раз складывает в переменную `post` следующий элемент из массива `posts` (начиная с нулевого):

The screenshot shows a web browser's developer console with a JavaScript file named `app.js` open. The code defines an array `posts` with two objects and a `for` loop that iterates over each object, logging its `text` property to the console. A breakpoint is set at line 18, which is highlighted in blue. The console shows the message `Paused on breakpoint`. The `Scope` panel on the right shows the current state of the variables: `post` is an object with `authorName: "Алиф"`, `created: 1588235339`, `id: 2`, and `text: "Как отправлять деньги из России"`. The `posts` array is also visible in the `Script` panel, showing two objects.

```
1 const posts = [
2   {
3     id: 2,
4     authorName: 'Алиф',
5     created: 1588235339,
6     text: 'Как отправлять деньги из России'
7   },
8   {
9     id: 1,
10    authorName: 'Алиф',
11    created: 1588166860,
12    text: 'alif mobi пулхоятонро аз вирус'
13  },
14 ];
15
16
17 for (const post of posts) {
18   console.log(post.text);
19 }
```

Paused on breakpoint

- Watch
- Breakpoints
 - ☒ app.js:18
console.log(post.text);
- Scope
 - Block
 - post:
 - authorName: "Алиф"
 - created: 1588235339
 - id: 2
 - text: "Как отправлять деньги из России"
 - [[Prototype]]: Object
 - Script
 - posts: (2) [{...}, {...}]
 - Global

Window



Цикл

Помимо цикла `for...of`, существуют циклы `for`, `for...in`, `while` и `do...while`. Мы не будем с ними работать по одной простой причине – есть гораздо более важные инструменты, которым мы должны научить вас пользоваться уже сейчас, поскольку это вам в значительной степени облегчит обучение на курсе по React.

Вы самостоятельно можете изучить работу с циклами на соответствующей странице [MDN](#). MDN – это большой портал, предназначенный для предоставления информации о современных веб-технологиях. Обязательно ознакомьтесь с ним.



Добавление/удаление



Добавление/удаление

Кроме перебора элементов, у нас есть ещё пара вопросов: как нам добавить элемент в массив и удалить из массива?

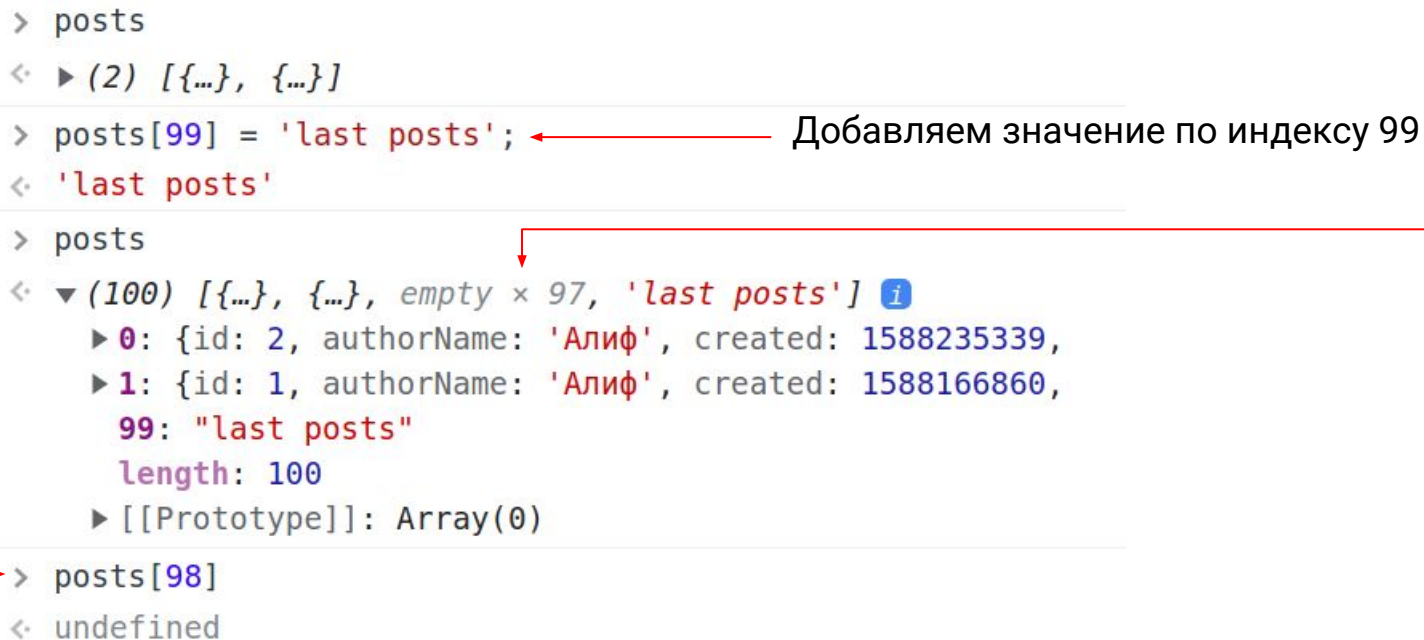
Давайте попробуем использовать те знания о свойствах объектов, которые у нас уже есть.



Свойства

JS, как вы помните, не запрещает добавлять в объект свойства:

```
> posts
< ▶ (2) [{...}, {...}]
> posts[99] = 'last posts';
< 'last posts'
> posts
< ▼ (100) [{...}, {...}, empty × 97, 'last posts'] ⓘ
  ▶ 0: {id: 2, authorName: 'Алиф', created: 1588235339,
  ▶ 1: {id: 1, authorName: 'Алиф', created: 1588166860,
    99: "last posts"
    length: 100
  ▶ [[Prototype]]: Array(0)
> posts[98]
< undefined
```



Мы сделали очень "нехорошую штуку". В нашем массиве появилась "дырка" на 96 элементов, при попытке обращения к которым, мы получим undefined.



Удаление

Примерно то же самое будет, если мы попробуем удалить элемент из массива с помощью `delete` (обновите страницу в браузере, чтобы те изменения, которые вы внесли до этого, исчезли):

```
> posts
< ▶ (2) [{...}, {...}]

> delete posts[1];
< true

> posts
< ▼ (2) [{...}, empty] ⓘ
  ▶ 0: {id: 2, authorName: 'Алиф', created: 1588235339,
    length: 2
  ▶ [[Prototype]]: Array(0)
```



Массивы

Таким образом, мы приходим к выводу, что нам нужен какой-то удобный инструмент, который позволит работать с массивами.

И он действительно есть: массивы – умные объекты, которые содержат достаточное количество полезных методов. Можно сказать, что большая часть вашей успешности в JS будет зависеть именно от того, как вы умеете работать с массивами.



Методы массива



Методы массива

Итак, мы говорили с вами о том, что объекты в качестве свойств могут содержать функции, которые умеют делать что-то полезное. Такие функции (когда они содержатся внутри свойств объекта) называются методами. Лучше всего о свойствах и методах читать на MDN (ещё лучше читать английскую версию MDN), но мы для просты посмотрим, как работать с русскоязычной.

Для того, чтобы что-то найти на MDN достаточно в Google вбить запрос вида:

The screenshot shows a Google search interface. The search bar contains the text 'mdn array'. Below the search bar, there are tabs for 'Все' (All), 'Видео' (Video), 'Новости' (News), 'Картинки' (Images), 'Покупки' (Shopping), 'Ещё' (More), and 'Инструменты' (Tools). The search results show approximately 1,600,000 results in 0.28 seconds. The top result is from mozilla.org, titled 'Array - JavaScript - MDN Web Docs'. Below the title, there is a table showing browser support for the Array object.

	Chrome	Edge
Array	Full support. Chrome1. Toggle history	Full support. Edge12. Toggle history
@@iterator	Full support. Chrome38. Toggle history	Full support. Edge12. Toggle history
@@species	Full support. Chrome51. Toggle history	Full support. Edge79. Toggle history

Показать ещё 46 строк



Методы массива

Методы

`Array.from()`

`Array.isArray()`

 `Array.observe()`

`Array.of()`

`Array.prototype.concat()`

`Array.prototype.copyWithIn()`

`Array.prototype.entries()`

`Array.prototype.every()`

`Array.prototype.fill()`

`Array.prototype.filter()`

`Array.prototype.find()`

`Array.prototype.findIndex()`

`Array.prototype.flat()`

`Array.prototype.flatMap()`

`Array.prototype.forEach()`

`Array.prototype.includes()`

`Array.prototype.indexOf()`

Нас будет интересовать левая боковая панель, в которой и представлен раздел Методы.

Интересовать нас будут только те методы, которые начинаются с `Array.prototype`. Что это значит, мы с вами поговорим на следующей лекции (это связано с тем самым непонятным свойством `[[Prototype]]`).

Для начала давайте попробуем найти замену циклам. В простейшем случае эта замена представляет собой метод `forEach`.



Методы массива

Страница документации достаточно подробно описывает то, как нужно использовать этот метод, но содержит ряд терминов и форматов записи, которые мы с вами не обсуждали:

```
arr.forEach(function callback(currentValue, index, array) {  
    //your iterator  
}, [thisArg]);
```

1. **callback** – вместо того, чтобы самим писать цикл, в котором всё делать, мы привыкаем жить по-другому: находим "умный" объект (такой как массив) и говорим, какую функцию нужно выполнить (в случае **forEach** – для каждого элемента)
2. **[thisArg]** – квадратные скобки означают необязательный параметр (а раз он необязательный – мы его не будем указывать, чуть позже разберёмся, что такое **thisArg**)



Методы массива

```
function printText(post) {  
  console.log(post.text);  
}
```

```
posts.forEach(printText);
```

Как это работает? Мы с вами говорили, что функция – это объект (фактически объявление функции создаёт имя). Это имя мы можем использовать не только для вызова функции, но и для передачи в качестве параметра в другую функцию.



Методы массива

Вот сейчас может показаться сложно, но всё решаемо, если использовать простые аналогии из жизни (главное, выбирать правильные аналогии). Например, вы пытаетесь дозвониться своему товарищу, а он не берёт телефон. Поэтому вы ему отправляете смс с текстом "перезвони мне".

Обратите внимание: "отправить смс" – это функция, в которую передано название другой функции – "перезвонить". И ваш товарищ действительно перезванивает тогда, когда готов выйти на связь.

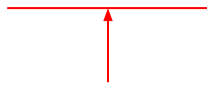


Коллбеки

Коллбеки (callbacks) – это одна из критичных вещей в понимании JS. Привыкните к тому, что когда вы работаете с набором объектов – вы начальник, а массив – это ответственный исполнитель. Вы говорите "что делать" (какую функцию вызывать), а он (массив) вызывает её для каждого элемента:

```
function printText(post) {  
  console.log(post.text);  
}
```

```
posts.forEach(printText);
```



говорим, что делать,
он сам это сделает для каждого
элемента



Параметры

Напомним вам, что в документации говорилось о том, что в функцию передаётся три параметра (а у нас только один, да и он называется по-другому):

```
arr.forEach(function callback(currentValue, index, array) {  
    //your iterator  
}, thisArg));
```

```
function printText(post) {  
    console.log(post.text);  
}
```

```
posts.forEach(printText);
```



Параметры

```
arr.forEach(function callback(currentValue, index, array) {  
    //your iterator  
}, [, thisArg]);
```

```
function printText(post) {  
|   console.log(post.text);  
}
```

```
posts.forEach(printText);
```

Ключевых момента здесь два:

1. Название роли не играет, играет роль только порядок (что параметр идёт первым в списке параметров)
2. Вы можете передавать больше параметров чем заявлено, они будут просто проигнорированы* (вспомните прошлую лекцию)

Примечание*: на самом деле JS оставит возможность до них добраться, но это не важно.



Анонимные и стрелочные функции



Имена для функций

```
function printText(post) {  
  console.log(post.text);  
}  
  
posts.forEach(printText);
```

Писать вот так, конечно, можно, но очень уж неудобно по двум причинам:

1. Очень сложно придумывать толковые имена*
2. Зачем придумывать целое имя ради того, чтобы использовать функцию всего один раз


Примечание*: по статистике, придумывать хорошие имена – самая сложная задача в программировании :-)



Анонимные функции

Для решения этой проблемы JS предлагает концепцию анонимных функций, т.е. функций, у которых нет имени. Что мы делаем? Просто удаляем имя у функции, а саму функцию помещаем в то место, где она используется:

```
function printText(post) {  
  console.log(post.text);  
}  
  
posts.forEach(printText);
```



```
posts.forEach(function(post) {  
  console.log(post.text);  
});
```

Здесь главное – это быть аккуратным и не запутаться в фигурных и круглых скобках. Поэтому возьмите за правило: если пишете открывающую скобку, то сразу пишете закрывающую для неё.



Стрелочные функции

Это не самая короткая запись, поскольку в JS ввели новый вид функций, который называется стрелочными. Помимо того, что они позволяют писать короче, они обладают ещё рядом особенностей, которые нас пока не будут интересовать.

Давайте пойдём по шагам.



Стрелочные функции

1. Слово `function` не пишется, вместо этого после круглых скобок ставится `=>` (fat arrow) – равно и символ больше:

```
posts.forEach(function(post) {  
  console.log(post.text);  
});
```



```
posts.forEach((post) => {  
  console.log(post.text);  
});
```

2. Если тело функции представляет из себя всего одну инструкцию, то фигурные скобки и `;` можно не писать (если бы ещё был `return`, то его тоже можно было бы опустить):

```
posts.forEach((post) => {  
  console.log(post.text);  
});
```



```
posts.forEach((post) => console.log(post.text));
```

3. Если параметр у функции всего один, то и круглые скобки вокруг параметра можно не писать:

```
posts.forEach((post) => console.log(post.text));
```



```
posts.forEach(post => console.log(post.text));
```



Стрелочные функции

В итоге всё свелось к одной строке. По началу это может показаться достаточно сложным, особенно если вы начинали изучение "классическим" способом через книжки и веб-сайты.

Но вы должны привыкать именно к этому способу, потому сейчас он наиболее распространённый. И у вас есть выбор: либо переучиться сейчас, либо тратить на это время позже.



ИТОГИ



Итоги

В этой лекции мы обсудили ключевую тему нашего курса, от понимания которой зависит всё остальное ваше обучение. Поэтому не поленитесь и перечитайте лекцию несколько раз: до тех пор, пока не усвоите все аналогии, приведённые в ней.

Мы также рекомендуем не терять времени зря и ознакомиться со всеми методами массива, представленными на странице документации MDN. Мы будем ещё детальнее разбираться со стрелочными функциями и методами на следующих лекциях. Сейчас же ключевое - вы должны начать их использовать.

Если у вас с этим возникнут сложности, не стесняйтесь, пишите в канал курса.



ДОМАШНЕЕ ЗАДАНИЕ



Орг.моменты

Практикум состоит из 8 обязательных занятий. Мы выкладываем новые занятия каждый понедельник в 14:00 (по Душанбе), кроме первой недели.

Каждое воскресенье в 23:59 (по Душанбе) дедлайн сдачи домашнего задания. Дедлайн – это предельный срок, до которого вы должны сдать ДЗ.

Если не успеете сдать в срок домашнее задание, тогда этот практикум будет для вас закончен и вы сможете зарегистрироваться на запуск следующего через несколько месяцев.

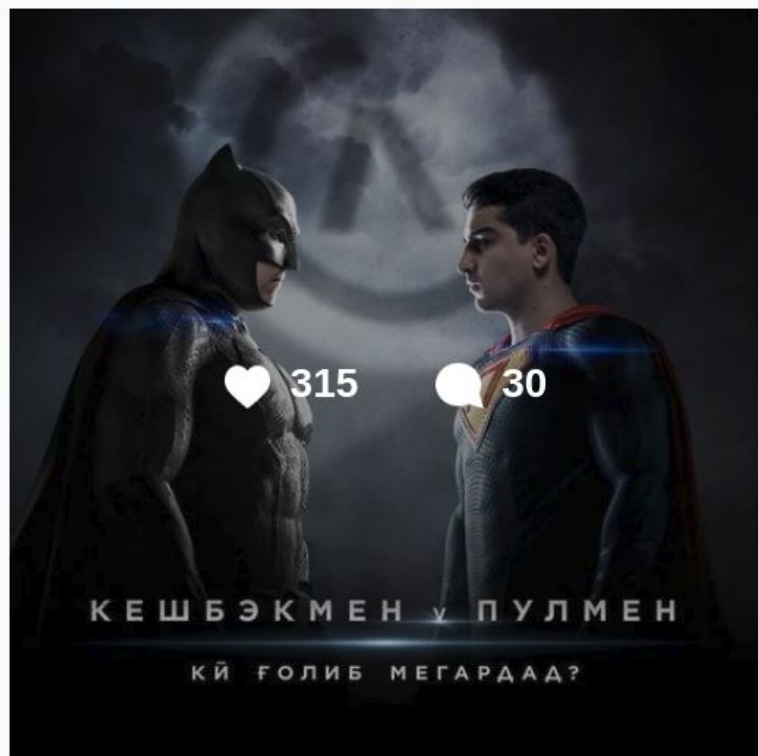
Все вопросы вы сможете задавать в [Телеграм канале](#).



ДЗ №1: Instamedia

Необходимо изучить страничку Alif в Instagram

https://www.instagram.com/p/CN9_JAaBk-9/ и правильно заполнить поля для следующего объекта:



Система будет проверять наличие следующих полей:

1. `id` – строка
2. `authorName` – строка
3. `likes` (количество лайков) – число
4. `comments` (количество комментариев) – число
5. `published` (дата публикации unix time) - число



ДЗ №1: Instamedia

В качестве **id** возьмите часть ссылки <https://www.instagram.com/p/ZZZZZZZZZZ/> (это не **id**, но мы пока будем считать, что **id**). Для того, чтобы получить ссылку, достаточно кликнуть на самом посте.

В качестве **authorName** берите название аккаунта (alifbank.tj).

В качестве количества лайков и комментариев берите данные со скриншота (с предыдущего слайда).

В качестве даты публикации берите **1619095140**.



ДЗ №1: Instamedia

Чтобы автоматизированная система смогла проверить вашу задачу, необходимо выполнить ряд требований:

1. Внутри архива должен быть каталог `instamedia`
2. Название объекта должно быть `post` (`const post = { ... };`)
3. Названия полей и их значения (а также типы) должны полностью соответствовать требуемым

```
1  const post = {  
2  |    /* здесь набор ваших полей */  
3  |  };  
4  
5  console.log(post);
```



ДЗ №2: Salom

На сайте представлен [набор партнёров](#), которые сотрудничают с Алиф Банком.

Вам нужно проанализировать представленные объекты и создать массив, в котором будут храниться объекты, следующих партнёров:



Тобон • 12 мес.
Отопление

0%



Charme • 3 мес.
Женская одежда

0%



Ақиқа • 6 мес.
Ювелирные изделия



Fosila • 24 мес.
Оргтехника



ДЗ №2: Salom

Какие поля должны быть у объектов:

1. `id` – строка, используйте из адреса ссылки
2. `url` – ссылка на страницу описания
3. `logoUrl` – ссылка на картинку (используйте <https://alif-skills.pro/media/id-партнёра.svg>), вместо id-партнёра подставьте `id` из п.1
4. `title` – название компании (как есть)
5. `category` – название категории (на русском)
6. `period` – срок рассрочки (в месяцах), число



ДЗ №2: Salom

Чтобы автоматизированная система смогла проверить вашу задачу, необходимо выполнить ряд требований:

1. Внутри архива должен быть каталог `salom`
2. Название массива должно быть `partners` (`const partners = [ваши элементы]`)
3. Массив должен содержать ровно 4 элемента в том порядке, в котором они указаны на скриншоте
4. Названия полей и их значения (а также типы) должны полностью соответствовать требуемым



ДЗ №3: Thanos

В Google была достаточно интересная шутка, которая заключалась в следующем: если вы вбивали в поиске слово Thanos, то в боковой панели появлялась вот такая "перчатка":



Танос

Персонаж



При нажатии на которую со страницы пропадало половина результатов поиска. Вам нужно сделать почти так же – а именно написать функцию, которая на вход принимает массив и возвращает новый массив, в котором содержатся только элементы с нечётными индексами из первого массива.



ДЗ №3: Thanos

Как это выглядело (а также другие спрятанные подобные "пасхалки") вы можете найти в ролике на Youtube: <https://www.youtube.com/watch?v=l4aiKKO0w8o>



ДЗ №3: Thanos

Для этого вам необходимо будет изучить [документацию на метод filter](#).

Вот так должна выглядеть ваша функция:

```
1  function filterByThanos(items) {  
2    |    return items.filter(/* сюда дописываете свой код */);  
3  }  
4  
5  const posts = [ /* перечисляете посты */ ];  
6  const filtered = filterByThanos(posts);  
7  
8  console.log(filtered);
```

Допишите код в участки, находящиеся между `/* */` (сами эти символы нужно удалить)



ДЗ №3: Thanos

Как это работает? Сначала вычисляется то, что стоит после `return` и только потом полученное значение возвращается из функции. Поскольку из метода `filter` возвращается массив, то именно он и кладётся в переменную `filtered` на 6-ой строке:

```
1  function filterByThanos(items) {  
2    |    return items.filter(/* сюда дописываете свой код */);  
3  }  
4  
5  const posts = [ /* перечисляете посты */ ];  
6  const filtered = filterByThanos(posts);  
7  
8  console.log(filtered);
```



ДЗ №3: Thanos

Чтобы автоматизированная система смогла проверить вашу задачу, необходимо выполнить ряд требований:

1. Внутри архива должен быть каталог `thanos`
2. Код должен соответствовать коду с предыдущего слайда (с учётом того, что вы допишете недостающие куски)
3. Функция `filterByThanos` должна работать только с индексами (ей ничего не нужно знать, какие поля есть в посте и посты ли она фильтрует)



ДЗ №3: Thanos

Подсказка: посмотрите на то, как работает оператор % (остаток от деления).



ДЗ №4: Notify

Достаточно часто в социальных сетях можно встретить функцию уведомления о новых постах. Мы немного её модифицируем и сделаем всё следующим образом: вы прочитаете документацию на методы [every](#) и [some](#) (выберите один из них) и напишите следующую функцию:

Что делает функция – она возвращает `true` только тогда, когда среди переданных в неё элементов хотя бы у одного в свойстве `read` стоит значение `false` (для нашего примера `notify` должно быть `true`).

```
1  function hasUnread(items) {  
2    /* ваш код */  
3  }  
4  
5  const posts = [  
6    {  
7      id: 1,  
8      read: true,  
9    },  
10   {  
11     id: 2,  
12     read: true,  
13   },  
14   {  
15     id: 3,  
16     read: false,  
17   },  
18 ];  
19  
20 const notify = hasUnread(posts);  
21 console.log(notify);
```



ДЗ №4: Notify

Подсказка: обратите внимание, что для пустого массива `hasUnread` всегда должна возвращать `false`.

Чтобы автоматизированная система смогла проверить вашу задачу, необходимо выполнить ряд требований:

1. Внутри архива должен быть каталог `notify`
2. Код должен соответствовать коду с предыдущего слайда (с учётом того, что вы допишете недостающие куски)
3. Функция `hasUnread` должна работать только со свойством `read`, ничего другого она использовать не должна



Спасибо за внимание

alif skills

2023г.

